

Algorithm keeps data safe

Luis G Uribe C, Caracas, Venezuela

Many embedded systems must regularly update multibyte data to EEPROM, flash memory, or a database server. This Design Idea presents a robust algorithm for this process that prevents data losses and inconsistencies due to program interruptions or power failures. You can avoid data losses by maintaining two separate memory areas, duplicating critical variables in each. Memory can be battery-backed RAM, magnetic disks, flash memory, or local or remote storage subsystems. You can use a simple FSM (finite-state machine) that uses three or four states with appropriate Gray coding to track

the status of each main variable and its mirrored value in the storage devices. The states run in a sequence that the software driver can use to write data to both the main and the backup variables. The driver sets and resets two variables, B0 and B1, as status bits.

B0 is the main variable status bit, and B1 is a mirror status bit. Both bits are recorded in the same storage medium as the data.

This algorithm prevents race condi-

tions and ambiguous situations. You can enter the power-up verification routine at any time, interrupting the main flow of the program without losing any data. Figure 1 shows the flow of the algorithm, which begins with a power-up routine. Table 1 identifies and defines each of the possible states for the variables. EDN

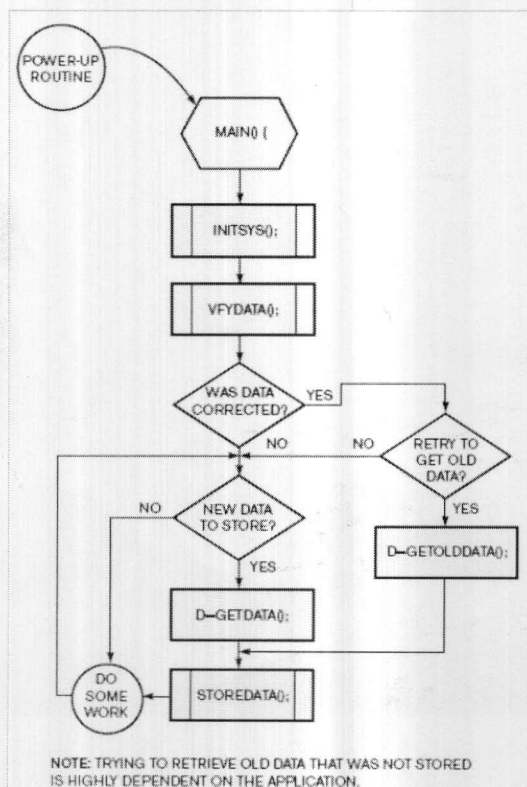


Figure 1 This algorithm starts with a power-up routine and then verifies that the data is properly stored. (For other algorithms, go to www.edn.com/100121dib.)

TABLE 1 POSSIBLE STATES OF VARIABLES

Bit state		Description
B0	B1	
0	0	Both variables are in steady state. If the power-up routine finds this state, it may assume that everything is fine; both values are properly stored in the main site and in the backup. Your program may read and use the data as needed. When it's time to modify a value, the driver sets a one at B0 to indicate a zero/one.
0	1	Data begins to write to the main variable. If the power-up routine finds this state upon verification, you can assume that the main variable is corrupt and decide how to update it from the mirroring site. When the driver finishes updating the main variable, it sets a one at B1 and keeps B0 untouched at one to indicate two ones.
1	1	Data has written properly to the main variable, and the driver soon begins the next state to duplicate main data into its backup storage. This state is transient to avoid changing bit values between zero/one and one/zero at once. Using this sequence, the two data repositories may be in different local or remote physical subsystems. If the power-up routine finds this state, it may assume that the mirror variable is corrupt, and it will proceed (next state) to update the mirror site from the main location. At any convenient time from now on, the finite-state machine may leave this state and enter the next one, one/zero, which means:
1	0	Data has properly written to the main variable state, and the driver now begins to duplicate the main data into backup storage (similar meaning to previous state, one/one). If the power-up routine finds this state, the verification routine may assume that the mirror variable is corrupt and update the mirror site from the main variable. When the driver finishes updating the mirroring variable, it clears B1, and the finite-state machine reverts to the first state, two zeros, in the sequence.